Dynamical Systems CW2

CID: 02044064

February 7, 2025

Q1

Let $f(x) = \lambda x + x^2 + x^3$ so that $\dot{x} = f(x)$. As seen in lectures, a fixed point x^* is stable if the derivative $f'(x^*) < 0$, and it is unstable if $f'(x^*) > 0$. x^* is half-stable if $f'(x^*) = 0$ and there exists $\epsilon \in \mathbb{R}$ such that $f'(x^* - \epsilon) > 0$ and $f'(x^* + \epsilon) < 0$.

Subcase B: $\lambda = 0$

Since $f'(x) = \lambda + 2x + 3x^2 = 2x + 3x^2$

$f'(0) = 0$

If $\epsilon > 0$ is small,

$f'(\epsilon) = 2\epsilon + 3\epsilon^2 > 0$ as $\epsilon > 0$

$f'(-\epsilon) = -2\epsilon + 3\epsilon^2 = \epsilon(3\epsilon - 2) < 0 \quad \forall \epsilon \in (0, \frac{2}{3})$

So $\forall \epsilon \in (0, \frac{2}{3})$, $f'(-\epsilon) < 0$, $f'(0) = 0$, $f'(\epsilon) > 0$

Thus, $x = 0$ is half-stable.

Subcase C: $0 < \lambda < \frac{1}{4}$

$f'(0) = \lambda > 0$ $\Rightarrow$ unstable

In the above 3 subcases, $\lambda < \frac{1}{4}$

$\lambda < \frac{1}{4} \Rightarrow -2(\lambda - \frac{1}{4}) > 0$ and $\sqrt{\frac{1}{4} - \lambda} > 0$

$\Rightarrow f'\left(-\frac{1}{2} - \sqrt{\frac{1}{4} - \lambda}\right) = -2(\lambda - \frac{1}{4}) + \sqrt{\frac{1}{4} - \lambda} > 0$

Thus, $-\frac{1}{2} - \sqrt{\frac{1}{4} - \lambda}$ is unstable for all $\lambda < \frac{1}{4}$.

However, $f'\left(-\frac{1}{2} + \sqrt{\frac{1}{4} - \lambda}\right) = -2(\lambda - \frac{1}{4}) - \sqrt{\frac{1}{4} - \lambda}$

$f'\left(-\frac{1}{2} + \sqrt{\frac{1}{4} - \lambda}\right) > 0$ $\Leftrightarrow$ $2(\frac{1}{4} - \lambda) < \sqrt{\frac{1}{4} - \lambda}$

unstable

$\frac{1}{4} - \lambda > 0$

$\Leftrightarrow$ $\sqrt{\frac{1}{4} - \lambda} > \frac{1}{2}$

$\Leftrightarrow$ $\frac{1}{4} < \frac{1}{4} - \lambda$

$\Leftrightarrow$ $\lambda < 0$

Replacing the $>$ signs in the previous $\iff$ statements with $=$ and $<$ signs, we also obtain

$f'\left(-\frac{1}{2}+\sqrt{\frac{1}{4}-\lambda}\right)=0 \iff \lambda=0$ (half-stable)

$f'\left(-\frac{1}{2}+\sqrt{\frac{1}{4}-\lambda}\right)<0 \iff \lambda>0$ (stable)

Case 1: $\lambda=\frac{1}{4}$

$f'(0)=\lambda=\frac{1}{4}>0$ $\therefore$ unstable

$f'\left(-\frac{1}{2}\pm\sqrt{\frac{1}{4}-\frac{1}{4}}\right)=f'\left(-\frac{1}{2}\right)=\frac{1}{4}+2\left(-\frac{1}{2}\right)+3\left(-\frac{1}{2}\right)^2=0$

$f''\left(-\frac{1}{2}\right)=2+6\left(-\frac{1}{2}\right)=-1<0$

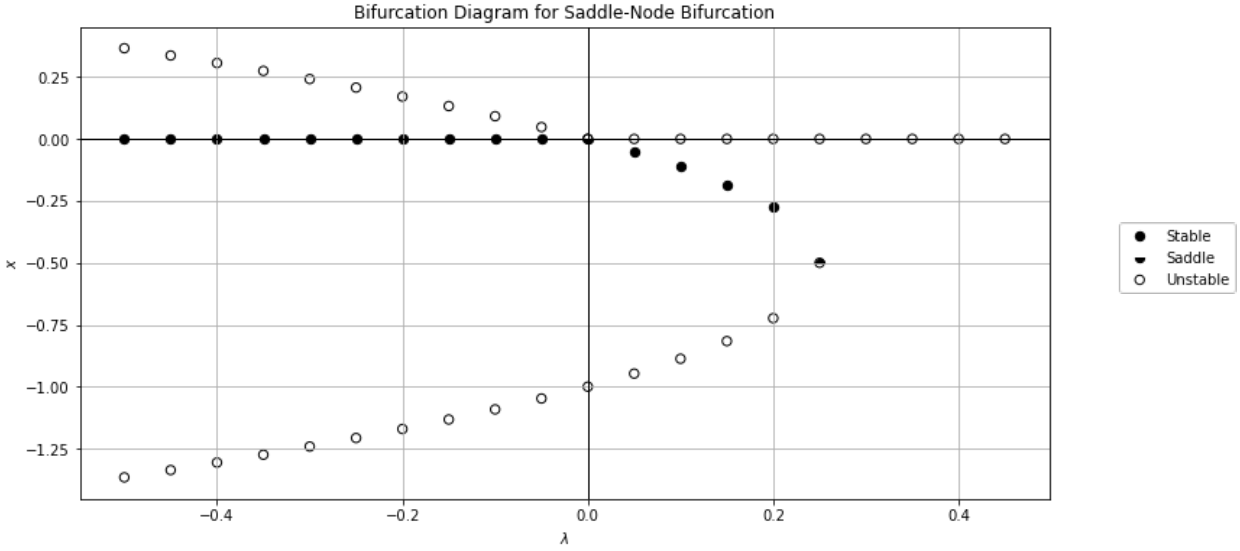
Case 2: $\lambda>\frac{1}{4}$ so $x=0$ is the only fixed point

$f'(0)=\lambda>\frac{1}{4}>0$ $\therefore$ unstable

Summary of results:

Define $\alpha=-1/2+\sqrt{1/4-\lambda}$ and $\beta=-1/2-\sqrt{1/4-\lambda}$

λ values	$x=0$ stability	$x=\alpha$ stability	$x=\beta$ stability
$\lambda<0$	Stable	Unstable	Unstable
$\lambda=0$	Half-stable	Half-stable	Unstable
$0<\lambda<\frac{1}{4}$	Unstable	Stable	Unstable
$\lambda=\frac{1}{4}$	Unstable	Half-stable	Half-stable
$\lambda>\frac{1}{4}$	Unstable	N/A	N/A



I have streamlined my code below by using the fact that whenever there are 3 distinct fixed points, the middle one is always stable, regardless of the parameter value.

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.markers import MarkerStyle


# Function to find the fixed points for the Saddle-Node Bifurcation
def saddle_node_bifurcation(lambda_vals):
    x_fixed_points = []
    for lam in lambda_vals:
        lam = round(lam, 2)  # round to 2 decimal places
        if lam < 1/4 and lam != 0:
            x1 = -1/2 + np.sqrt(1/4-lam)
            x2 = -x1 - 1
            x3 = 0
            x_fixed_points.append([x2, x3, x1])
        elif lam == 1/4:
            x_fixed_points.append([0, -1/2])
        else:
            x_fixed_points.append([0])
    return x_fixed_points


# Generate lambda values
lambda_vals = np.arange(-0.5, 0.5, 0.05)

# Compute fixed points
x_fixed_points = saddle_node_bifurcation(lambda_vals)

# Plotting the bifurcation diagram
plt.figure(figsize=(12, 6))


for i, lam in enumerate(lambda_vals):
    fixed_points = x_fixed_points[i]

    if len(fixed_points) == 3:

        # Stable fixed point in the middle of the 3
        middle_fixed_point = sorted(fixed_points)[1]
        plt.scatter(lam, middle_fixed_point, color='black', s=40, edgecolor='black',
        ↪  facecolor='black')

        # Unstable points
        for point in sorted(fixed_points)[::2]:
            plt.scatter(lam, point, color='black', s=40, edgecolor='black',
            ↪  facecolor='none')

    elif len(fixed_points) == 2:

        # Saddle points with half-filled circles
        plt.scatter(0, 0, color='black', s=40, marker=MarkerStyle("o",
        ↪  fillstyle="bottom"), label="Saddle")
```

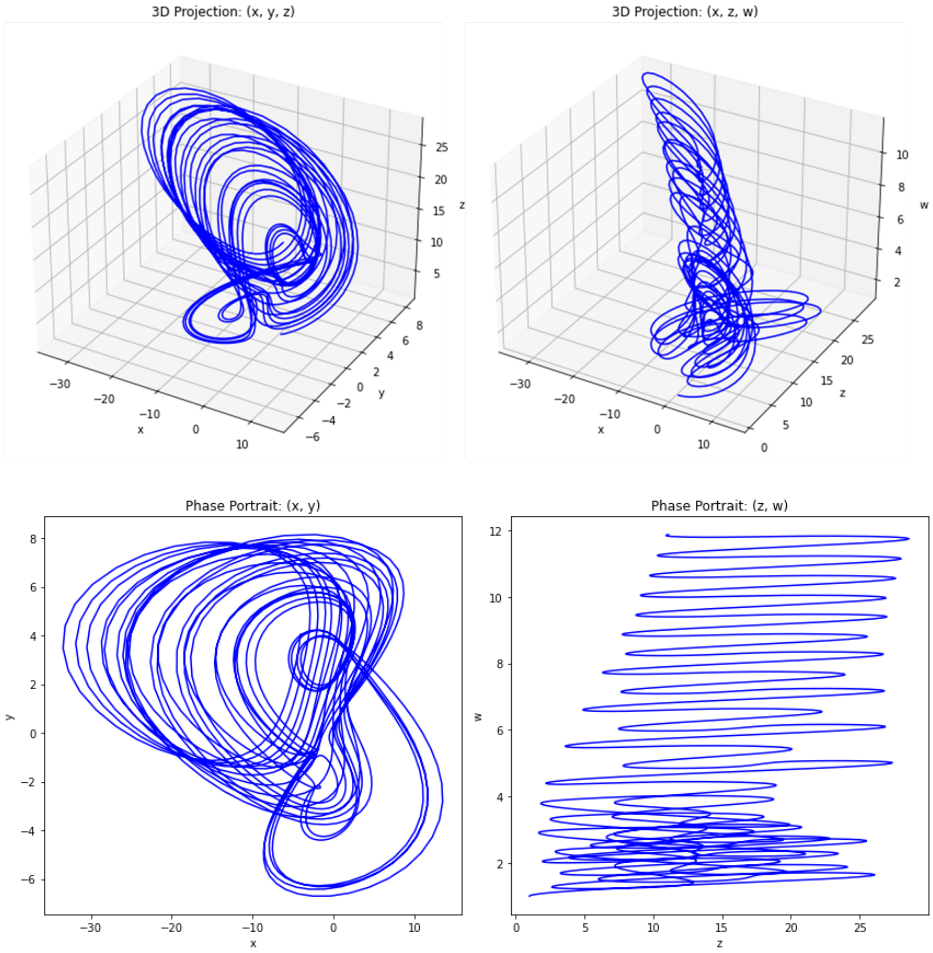
```
        plt.scatter(0, -1, color='black', s=40, edgecolor='black', facecolor='none')
        plt.scatter(1/4, -1/2, color='black', s=40, marker=MarkerStyle("o",
        ↪  fillstyle="top"))
        plt.scatter(1/4, -1/2, color='black', s=40, edgecolor='black', facecolor='none')
        plt.scatter(1/4, 0, color='black', s=40, edgecolor='black', facecolor='none')

    elif len(fixed_points) == 1:
        plt.scatter(lam, fixed_points[0], color='black', s=40, edgecolor='black',
        ↪  facecolor='none', label="Unstable")


# Add legend for different stabilities
legend_handles = [
    plt.Line2D([0], [0], marker='o', color='black', linestyle='None', label='Stable'),
    plt.Line2D([0], [0], marker='o', color='black', markerfacecolor='none',
    ↪  linestyle='None', label='Unstable'),
    plt.Line2D([0], [0], marker=MarkerStyle("o", fillstyle="top"), color='black',
    ↪  linestyle='None', label='Saddle')
]

# Formatting
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.title('Bifurcation Diagram for Saddle-Node Bifurcation')
plt.xlabel(r'$\lambda$')
plt.ylabel(r'$x$')
plt.legend(handles=legend_handles, loc='upper right')
plt.grid(True)
plt.show()
```

Q2



Plotting code:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Define the system of equations
def system(t, state, a, b, c, d, e):
    x, y, z, w = state
    dxdt = -a*x + b*y - y*z
    dydt = x + c*w**2 + e
    dzdt = y**2 - z
    dwdt = d*y
    return [dxdt, dydt, dzdt, dwdt]

# Parameters
a = 2.0
b = 10.0
c = 0.1
d = 0.1
e = 1.0

```

```
# Initial conditions
initial_conditions = [1.0, 1.0, 1.0, 1.0]

# Time span and evaluation points
time_span = (0, 100)
time_eval = np.linspace(0, 100, 2000)

# Solve the system using solve_ivp
sol = solve_ivp(system, time_span, initial_conditions, args=(a, b, c, d, e),
↪  t_eval=time_eval)

# Extract the solution
t = sol.t
x, y, z, w = sol.y  # Unpacking all state variables

# Plot 3D projections
fig = plt.figure(figsize=(12, 6))

# 3D Plot (x, y, z)
ax1 = fig.add_subplot(121, projection='3d')
ax1.plot(x, y, z, color='b')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')
ax1.set_title('3D Projection: (x, y, z)')

# 3D Plot (x, z, w)
ax2 = fig.add_subplot(122, projection='3d')
ax2.plot(x, z, w, color='b')
ax2.set_xlabel('x')
ax2.set_ylabel('z')
ax2.set_zlabel('w')
ax2.set_title('3D Projection: (x, z, w)')

plt.tight_layout()
plt.show()

# Plot 2D Phase Portraits
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Phase portrait (x, y)
ax[0].plot(x, y, color='b')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y')
ax[0].set_title('Phase Portrait: (x, y)')

# Phase portrait (z, w)
ax[1].plot(z, w, color='b')
ax[1].set_xlabel('z')
ax[1].set_ylabel('w')
ax[1].set_title('Phase Portrait: (z, w)')

plt.tight_layout()
plt.show()
```

Code for chaos analysis:

```
from tqdm import tqdm

# Gram-Schmidt reorthogonalization function
def gram_schmidt(vectors):
    dim = vectors.shape[1]
    ortho_vectors = np.copy(vectors)
    norms = np.zeros(dim)

    for i in range(dim):
        for j in range(i):
            proj = np.dot(ortho_vectors[:, j], ortho_vectors[:, i]) * ortho_vectors[:, j]
            ortho_vectors[:, i] -= proj
        norms[i] = np.linalg.norm(ortho_vectors[:, i])
        ortho_vectors[:, i] /= norms[i]

    return ortho_vectors, norms

# Lyapunov spectrum calculation
def lyap_exp(f, dim, params, t_span, t_step, dt, x_0, transient=100):
    t_start, t_end = t_span
    timesteps = int(round((t_end - t_start) / t_step))
    y = np.hstack((x_0, np.eye(dim).flatten()))  # State + tangent vectors
    cum = np.zeros(dim)
    t = t_start

    # Integration and reorthogonalization loop
    for _ in range(timesteps):
        sol = solve_ivp(f, [t, t + t_step], y, args=(params,), max_step=dt)
        y = sol.y[:, -1]

        # Extract tangent vectors and reorthogonalize using Gram-Schmidt
        tangent_vectors = y[dim:].reshape(dim, dim).T
        ortho_vectors, norms = gram_schmidt(tangent_vectors)
        y[dim:] = ortho_vectors.T.flatten()

        # Accumulate logarithms of norms
        if t > transient:
            cum += np.log(norms)

        t += t_step

    # Return average Lyapunov exponents
    return cum / (t_end - t_start - transient)


# 4D system
def system(t, X, params):
    x, y, z, w = X[:4]
    Y = X[4:].reshape(4, 4).T
    a, b, c, d, e = params
    f = np.zeros(20)
    f[:4] = [-a*x + b*y - y*z, x + c*w**2 + e, y**2 - z, d*y]
    Jac = np.array([[-a, b-z, -y, 0],
```

```
                    [1, 0, 0, 2*c*w],
                    [0, 2*y, -1, 0],
                    [0, d, 0, 0]])
    f[4:] = (Jac @ Y).T.flatten()
    return f


# parameters (that haven't already been defined)
t_span = (0, 100)  # Time span for integration
t_step = 0.1  # Integration step size
dt = 0.001  # Maximum step size
transient = 10  # Transient time to discard

# Storage for Lyapunov spectra
lyapunov_spectra = []

params = (a, b, c, d, e)
L = lyap_exp(system, dim=4, t_span=t_span, t_step=t_step, dt=dt, x_0=initial_conditions,
↪  params=params, transient=transient)
lyapunov_spectra.append(L)

# Convert results to a numpy array for easy manipulation
lyapunov_spectra = np.array(lyapunov_spectra)

print(lyapunov_spectra)
```

The Lyapunov spectrum is:
0.09756855 -0.02313418 -0.38170827 -2.68939276
The largest Lyapunov exponent is positive so the system is in a chaotic regime.

Q3

a) Claim: (0, 0) is the only fixed point.

Proof of claim:

(0, 0) is a fixed point because when x = 0 and y = 0, x dot = 0 and y dot = 0.

To show that (0, 0) is the unique fixed point, we show that (x dot, y dot) = (0, 0) implies (x, y) = (0, 0).

$\dot{x} = 0 \Rightarrow x\dot{x} = r(x^2 - xy) - x^2(x^2+y^2) = 0 \quad (1)$

$\dot{y} = 0 \Rightarrow y\dot{y} = xy + r(y^2 - y^2(x^2+y^2) = 0 \quad (2)$

$(1) + (2)$: $r(x^2+y^2) - (x^2+y^2)(x^2+y^2) = 0$

$\Rightarrow (r - (x^2+y^2))(x^2+y^2) = 0$

$\Rightarrow r = x^2+y^2$ or $x^2+y^2 = 0$

If $r = x^2+y^2$, $-\dot{x} = -(rx - y - rx) = y = 0$

and $\dot{y} = x + ry - ry = x = 0$

If $x^2+y^2 = 0$, then $x = y = 0$ as $x, y \in \mathbb{R}$

So in both cases, x = y = 0.

b)

Left multiply both sides by the inverse matrix:

Thus, the evolution of r is governed by